IN THE CLAIMS:

1-19   (Canceled)

20.   (Withdrawn) A method to preserve the state and data of the host OS in a mobile device where the boot loader manipulates the memory protection bits for pages of the memory if both OS use the MMU (Memory Management Unit) of the mobile device which supports memory protection; changing attributes of all memories used in the original OS (data and code) from read/write to read-only (not write-able) or even not accessible so that those memories of the host OS can no longer be modified after switching to guest OS.

21.   (Withdrawn) The method of claim 20, wherein the memory protection works on physical memory space.

22.   (Withdrawn) The method of claim 20, wherein the memory protection works on virtual memory space and both OS share the same virtual memory.

23.   (Withdrawn) A method to preserve the state and data of the host OS in a mobile device to move/copy data of the host OS to the top of the system memories and faked a reduced memory environment to the guest OS. When the guest OS starts, it will think that it is running on a reduced memory device and will never touch memories above the told highest available memory that are used to backup the host OS data, thus the data and state of the host OS is preserved.

24.   (Withdrawn) The method of claim 23, wherein the boot loader writes to a special register to fake the available memories to guest OS.

25.   (Withdrawn) The method of claim 23, wherein the guest OS changed its memory detection module to assume the faked maximum memory size.

26.   (Withdrawn) The method of claim 23, wherein the guest OS can have its own memory management module or virtual memory space as long as it use only memories it detected on booting.

27.   (Withdrawn) The method of claim 23, wherein the memories of mobile devices are divided into several banks or memory addresses are not continuous and the booting system fake that one memory bank is no longer available.

28.   (Withdrawn) A method to use a special memory device driver loaded during the initialization of the guest OS to claim memories of the host OS. The special memory device driver can claim those memories currently used by the host OS (including data, code and runtime) and keep its content from being modified by other programs of the guest OS until guest OS exit.

29.   (Withdrawn) The method of claim 28, wherein the special memory device driver mark the memories as inaccessible to the other part of the guest OS.

30.   (Withdrawn) The method of claim 28, wherein the special memory device driver is a kernel mode driver in guest OS that prevent the memory from being used by both kernel and user space processes.

31.   (Withdrawn) The method of claim 28, wherein if both OS need to use a same memory location, it may temporarily move the memory of the host OS to free areas and move back upon termination of the guest OS.

32.   (Withdrawn) A method to preserve the state and data of the host OS in a mobile device by backing up the current state of the host OS to an external memory cards. The process is: boot loader first save the whole OS memory image into external memory device and an exit code restore the host OS from the memory image from the external card upon retuning from the guest OS.

33.   (Withdrawn) The method of claim 32, wherein the memory cards are Compact Flash (CF) memory card or Secure Digital (SD) memory card or MMC card.

34.   (Withdrawn) The method of claim 32, wherein the whole memory images can be saved to memory cards as a backup point and being restored later.

35.   (Withdrawn) The method of claim 32, wherein the saved memory images can transferred to another mobile device, emulator or networks for mirroring purpose.

1   36.   (Currently Amended)   A method for an original operating system (a
2   host OS) in a mobile device that supports a memory protection mechanism to run
3   another operating system (a guest OS) within the same memory space of said
4   host OS while preserving the current state of said host OS in memory throughout
5   the execution of said guest OS, comprising the steps of:
6   said mobile device running said host OS;
7   said host OS starting said guest OS through a launcher;
8   said launcher going [[go]] through said memory protection mechanism to
9   mark memory blocks currently used by said host OS as protected from said
10   guest OS;
11   said launcher launching said guest OS;
12   said guest OS running, accessing only memory blocks that have not been
13   marked as protected so that said memory blocks marked as protected are
14   preserved throughout the execution of said guest OS;
15   said guest OS finishing finish running through an exit-code;
16   said exit-code restoring restores the state of said host OS by reverting
17   said protected memory blocks; and
18   said host OS resuming its operation.

1   37.   (Currently Amended)   The method of claim 36, ~~wherein said memory~~
2   ~~protection mechanism is~~ achieved through a memory management unit (MMU)
3   that <u>allows or disallows</u> ~~allow or disallow~~ a program to access particular memory
4   addresses.

1   38.   (Currently Amended)   The method of claim 36, wherein said launcher
2   <u>going</u> [[go]] through said memory protection mechanism to mark memory blocks
3   currently used by said host OS as protected from <u>said</u> guest OS further
4   <u>comprises the</u> ~~comprising~~ steps of:
5   moving memory blocks currently used by <u>said</u> host OS whose address
6   range is needed by said guest OS to a free memory location in the <u>mobile</u>
7   <u>device</u>; <u>and</u>
8   marking said free memory blocks as protected;
9   and wherein <u>restoring</u> ~~restores~~ the state of said host OS by reverting said
10   protected memory blocks further <u>comprises the</u> ~~comprising~~ steps of:
11   restoring said free memory blocks to said memory blocks whose address
12   range is needed by said guest OS; <u>and</u>
13   reverting said free memory blocks from protected.

1   39.   (Currently Amended)   The method of claim 36, <u>further comprising</u>
2   <u>starting</u> ~~to start~~ a second guest OS from said guest OS, ~~further~~ <u>wherein starting</u>
3   <u>the second guest OS comprises</u> ~~comprising~~ <u>the</u> steps of:
4   said guest OS acting as a host OS to the second guest OS;
5   said second guest OS acting as a guest OS; <u>and</u>
6   <u>repeating the</u> ~~Repeating~~ steps in claim 36 to start said second guest OS
7   from said guest OS.

1   40.   (Currently Amended)   A method for an original operating system (a
2   host OS) of a mobile device to start another operating system (a guest OS) while
3   keeping the running state and data of said host OS in memory throughout the
4   execution of said guest OS, comprising the steps of:

5   said mobile device running said host OS;

6   said host OS starting said guest OS through a launcher;

7   said launcher moving memory blocks in lower address space that are

8 currently used by said host OS to free memory blocks in upper address space,

9 and eventually preserving current state and data of said host OS to the upper

10 address space through the following steps:

11   for each of said memory blocks in lower address <u>space</u> currently

12 used by said host OS, <u>finding</u> ~~find~~ a free memory block in the upper address

13 space; and

14   <u>moving</u> ~~move~~ said used memory block in lower address space to

15 said free memory block in upper address space;

16   said launcher identifying a memory address location where <u>memory</u>

17 <u>addresses</u> ~~memories~~ above said location contains host OS data and memory

18 addresses below said memory address location <u>are</u> ~~to be~~ free to <u>be used</u> ~~use~~ by

19 said guest OS;

20   said launcher launching said guest OS[[,]] <u>by</u> passing <u>said</u> memory

21 address location as a reduced memory size to said guest OS;

22   said guest OS running in said <u>reduced in size</u> ~~reduce-sized~~ memory space

23 and <u>leaving</u> ~~leave~~ memory space higher than said memory location untouched;

24   said guest OS <u>finishing</u> ~~finish~~ running through an exit-code;

25   said exit-code <u>restoring</u> ~~restores~~ the state and data of said host OS by

26 reverting each of said memory blocks in <u>said</u> lower address <u>space</u> from each of

27 said free memory blocks in <u>said</u> upper address <u>space</u>; <u>and</u>

28   said host OS resuming operation.

1   41.   (Currently Amended)   The method of claim 40, wherein said launcher

2 launching said guest OS[[,]] <u>by</u> passing memory address location as a reduced

3 memory size to said guest OS[[;]] further <u>comprises</u> ~~comprising~~:

4   a memory detection module of said guest OS <u>that</u> uses said <u>reduced in</u>

5 <u>size</u> ~~reduce-sized~~ memory space as the total available memory of the mobile

6 device instead of the real memory size of the mobile device.

42. (Currently Amended)   The method of claim 40, wherein said launcher launching said guest OS[[,]] by passing memory address location as a reduced memory size to said guest OS[[;]] further comprises comprising:
     said launcher modifying a system register in said mobile device to report said reduced memory size as total available memory memories to said guest OS.

43. (Currently Amended)   The method of claim 40, wherein memories of said mobile device are divided into multiple memory banks, and said launcher launching said guest OS[[,]] by passing memory address location as a reduced memory size to said guest OS[[;]] further comprises comprising:
     generating Generating said reduced memory size by disabling one or more of said memory banks to make them no longer available to said guest OS.

44. (Currently Amended)   The method of claim 40, wherein input output (IO) states[[,]] and registers of said mobile device within said host OS are preserved and later restored into said free memory block in upper address space.

45. (Currently Amended)   A method for an original operating system (a host OS) of a mobile device to start another operating system (a guest OS) within the same memory space of said host OS while keeping the running state of said host OS throughout the execution of said guest OS in place, comprising the steps of:
     said mobile device running said host OS;
     said host OS starting said guest OS through a launcher;
     said launcher launching said guest OS, passing a list of memory addresses of currently used memory of said host OS to a memory device driver in said guest OS during initialization of said guest OS;
     said memory device driver of said guest OS claiming said list of memory addresses and keeping them from being modified by any other part of said guest OS during the execution of said guest OS;

14   said guest OS running in the same memory space of said host OS, with

15   memories used by said host OS being claimed and protected by under said

16   memory device driver;

17   said guest OS finishing finish running through an exit-code;

18   said exit-code restoring restores the state of said host OS by releasing

19   said list of memory addresses from said device driver; and

20   said host OS resuming operation.

1   46.   (Currently Amended)   The method of claim 45, wherein said

2   list of memory addresses used by host OS includes include memory addresses

3   only in dynamic runtime memory of said host OS and not memories.

1   47.   (Currently Amended)   The method of claim 45, wherein said

2   list of memory addresses used by host OS also includes include the current input

3   output (IO) states and current registers of the mobile device.

1   48.   (Canceled)

1   49.   (Canceled)

1   50.   (Currently Amended)   A method of packaging an image of a guest

2   OS inside a native application of a host OS to allow in-place execution of the said

3   guest OS image in order to reduce memory usage where the format of said

4   native application contains multiple un-continuous data chunks in memory space,

5   comprising the steps of:

6   compiling an image of said guest OS into multiple code segments;

7   appending each of said multiple code segments with a jump table

8   containing multiple jump addresses to be used to point to others of the the other

9   code segments;

8

10   for each of a plurality of inter-segment jump instructions in each of said

11 code segments, linking each of said plurality of inter-segment jump instructions to

12 point to an entry of said jump table;

13   preparing a native application for said host OS with a startup code and

14 multiple data chunks;

15   each of said multiple data chunks wrapping each of said code segments of

16 said guest OS plus each of said jump tables;

17   said host OS launching said guest OS by running said native application;

18   said startup code in said native application looping through entries of each

19 of said jump table entries to fill in the current memory address of each

20 corresponding code segment segments;

21   each of said code segments segment executing a jump instruction to

22 invoke codes in the other another code segments segment;

23   said executed jump instruction getting a real address of said other another

24 code segment from said jump tables;

25   said executed jump instruction successfully jumping to the other code

26 segments segment wrapped by a corresponding said another data chunk; and

27   wherein said image of guest OS executes now executing in place without

28 the need to extract all the code segments from said native application and

29 rearrange them in a continuous memory location.

1   51.   (Currently Amended)   The method of claim 50, wherein said

2   native application in said host OS is a database file with multiple data records or

3   a Palm PDB file with multiple Palm DB records.

1   52.   (Currently Amended)   The method of claim 50, wherein said native

2   application in said host OS is a Palm PDB file with multiple Palm DB records and

3   said host OS is a Palm OS.

1   53.   (Currently Amended)   The method of claim 50, wherein said image of

2   said the guest OS code requires one sequential memory address, said native

3   application of said host OS contains multiple data chunks that have has a

4   maximum chunk size,[[;]] and compiling an image of said guest OS into multiple

5   code segments further comprises comprising the steps of:

6   compiling said image of said guest OS into one code segment with

7   sequential memory addresses address;

8   re-arranging instructions in said image of said guest OS to reserve spaces

9   to be used for said jump tables on each of a plurality of said chunk-size

10   boundaries by inserting jump instructions to bypass by-pass those spaces; and

11   splitting said one code segment into multiple code segments on each of

12   said plurality of chunk-size said chunk-size boundaries.[[;]]

1   54.   (Currently Amended)   The method of claim 50, wherein the size of

2   said guest OS image exceeds the maximum size limit of said native application

3   of said host OS, further comprising the steps of:

4   splitting said image of said guest OS into multiple pieces;

5   compiling each of said multiple pieces piece of said image of said guest

6   OS into multiple code segments; and

7   packaging each of said multiple pieces piece of said image of guest OS

8   into a native application of said host OS.[[;]]

10